

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2000-222228

(43)Date of publication of application : 11.08.2000

(51)Int.Cl.

G06F 9/46

G06F 11/30

G06F 15/00

(21)Application number : 11-021249

(71)Applicant : HITACHI LTD

(22)Date of filing : 29.01.1999

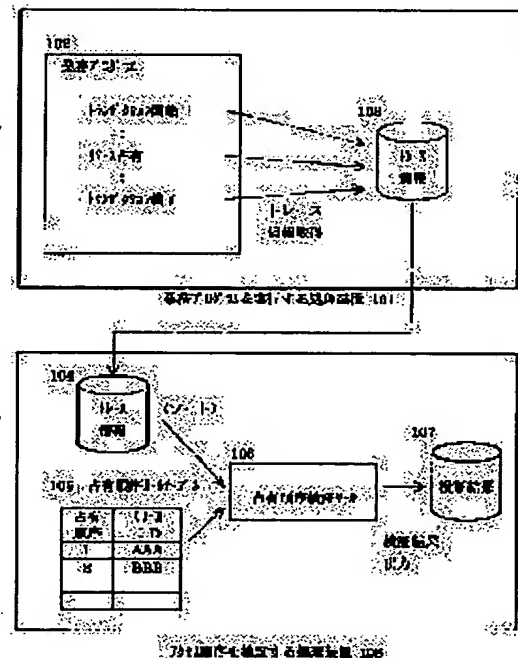
(72)Inventor : IJIMA MINORU

(54) DEADLOCK PREVENTING METHOD BY VERIFICATION OF RESOURCE OCCUPATION ORDER

(57)Abstract:

PROBLEM TO BE SOLVED: To verify the access order of resources in a test process and to prevent a deadlock in actual operation by tracing the history of resource occupation and inputting the result to an access order verification tool of differently generated resources.

SOLUTION: A device comprises a processor 101 which conducts a system test of an operation program and a processor 108 which verifies access order. The operation program 102 codes a process for obtaining trace information 103 right after a process for occupying a resource. The processor 108 which verifies occupation order inputs trace information 104 and an occupation order rule table 105 to the occupation order verifying tool 106 to obtain a verification result 107. The occupation order of resources can be confirmed and order illegality can mechanically be detected. Further, the resource access order verifying tool can be generated independently of a program main body and exerts no influence on the development process of the program main body.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2000-222228

(P 2 0 0 0 - 2 2 2 2 2 8 A)

(43) 公開日 平成12年8月11日(2000.8.11)

(51) Int. Cl. ⁷	識別記号	F I	テーマコード (参考)
G06F 9/46	340	G06F 9/46	G 5B042
11/30	305	11/30	G 5B085
15/00	320	15/00	K 5B098

審査請求 未請求 請求項の数 1 O L (全 5 頁)

(21) 出願番号 特願平11-21249

(22) 出願日 平成11年1月29日(1999.1.29)

(71) 出願人 000005108

株式会社日立製作所

東京都千代田区神田駿河台四丁目6番地

(72) 発明者 飯島 実

神奈川県川崎市幸区鹿島田890番地 株式

会社日立製作所情報システム事業部内

(74) 代理人 100068504

弁理士 小川 勝男

Fターム(参考) 5B042 GA23 GB01 HH20 HH30 MA14

MC17 MC22 NN01

5B085 AC08 BA07

5B098 GA04 GB01 GD01 GD16 GD24

JJ07

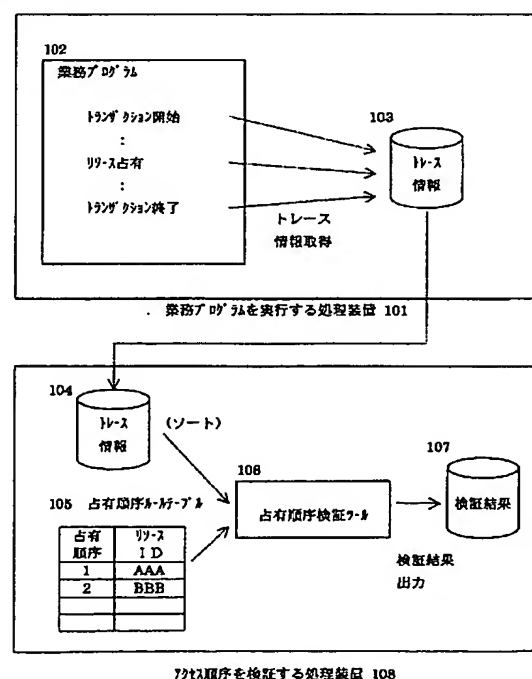
(54) 【発明の名称】 リソース占有順序の検証によるデッドロック防止方法

(57) 【要約】

【課題】 オンラインシステムにおいては、資源アクセス順序の不正によりデッドロックが発生する可能性があるが、同時処理の一瞬のタイミングでしか発生しない場合が多いため、実機を使用してシステムテストを行っても必ずしもそれを検出できるとは限らない。

【解決手段】 実機を使用したテストで取得したトレース情報から、リソースの占有順序を検証することにより、デッドロックの要因を抽出できる。

図 1



【特許請求の範囲】

【請求項 1】 1 トランザクションで多数のリソースへアクセスするマルチタスクのオンライントランザクション処理システムにおいて、実行時のトレースファイルからトランザクションが占有するリソースの占有順序規則を検証し、順序不正を摘出することにより、デッドロック要因の内在を事前に検知することを特徴とする排他制御におけるデッドロック防止方法。

【発明の詳細な説明】

【 0 0 0 1 】

【発明の属する技術分野】 本発明はオンライントランザクション処理システムのように 1 つのトランザクションで複数のリソースを占有要求するマルチタスクの環境に適用する。

【 0 0 0 2 】

【従来の技術】 特開平 7 - 1 2 1 3 8 7 号公報「排他制御におけるデッドロック防止方法」では実行時に O C P 側でリソースアクセス順序をチェックする方式である。

【 0 0 0 3 】 この方式は、データベースアクセスを含め、全てのリソースアクセスを 1 つの O C P で管理していることが前提になる。

【 0 0 0 4 】 近年、国際的標準化や流通パッケージソフトの普及により、O C P ソフトやデータベースソフトはパッケージソフトとして一般に市販されているものを組み合わせて適用するのが普通であり、上記の方式は適用できない状況にある。

【 0 0 0 5 】 業務プログラムの中でリソースアクセス順序をチェックするのは開発時の負担が大きく、本来の処理に対する悪影響が発生する可能性も少なくない。

【 0 0 0 6 】

【発明が解決しようとする課題】 デッドロック防止のためにはリソース占有順序を決め、その順序を厳守して、プログラムを作成しなければならない。

【 0 0 0 7 】 しかし、ヒューマンエラー等により、その順序が守られずに、デッドロック発生の原因になるという問題がある。

【 0 0 0 8 】 本発明の目的は、テスト工程においてリソースのアクセス順序を検証し、実運用でのデッドロック発生を防止することである。

【 0 0 0 9 】

【課題を解決するための手段】 オンラインシステムのプログラムではテスト結果の確認や障害発生時の調査を容易にするためにトレース情報をファイルに出力するのが一般的である。

【 0 0 1 0 】 この中に資源占有の履歴をトレースし、別に作成したリソースのアクセス順序検証ツールの入力とすることにより、リソースアクセス順序不正を摘出することができる。

【 0 0 1 1 】

【発明の実施の形態】 本発明の実施例を図面を参照して

説明する。

【 0 0 1 2 】 まず、全体システム構成図を図 1 に示す。

【 0 0 1 3 】 業務プログラムのシステムテストを行う処理装置 1 0 1 とアクセス順序を検証する処理装置 1 0 8 から構成される。業務プログラム 1 0 2 はリソースを占有する処理の直後にトレース情報 1 0 3 を取得する処理をコーディングしておく。占有順序を検証する処理装置 1 0 8 では、トレース情報 1 0 4 と占有順序ルールテーブル 1 0 5 を占有順序検証ツール 1 0 6 の入力として、
10 検証結果 1 0 7 を得る。

【 0 0 1 4 】 占有順序ルールテーブル 1 0 5 は図 3 に示すように、リソース I D (3 0 2) ごとに占有順序 (3 0 1) を規定したものである。

【 0 0 1 5 】 次に、トレース取得情報の例を図 4 に示す。

【 0 0 1 6 】 トランザクション I D (4 0 1) はトランザクション毎に採番される識別子で、時刻 (4 0 2) は占有要求の実時刻である。

【 0 0 1 7 】 トレースファイルには同時に動作する複数のトランザクションのトレース情報が混合して出力される。占有順序の検証はトランザクション I D ごとに行う必要があるため、検証の入力としてはトランザクション I D (4 0 1) と時刻 (4 0 2) をキーにしてソートしたものを使用する。

【 0 0 1 8 】 ソート後はトランザクション I D (4 1 1) ごとに検証単位 (4 1 0) となる。

【 0 0 1 9 】 モジュール名 (4 0 3) 、関数名 (4 0 4) 、行番号 (4 0 5) は業務プログラム内のリソース占有箇所を認識するための情報である。

30 【 0 0 2 0 】 リソース I D (4 0 6) は占有リソースを示す。トランザクション I D (4 0 1) が同一のトレース情報について、このリソース I D (4 0 6) は占有順序ルールテーブル 1 0 5 で規定された順序になっていることを検証しなければならない。

【 0 0 2 1 】 次に、図 2 により、占有順序検証ツールの処理フロー例を説明する。

【 0 0 2 2 】 (ステップ 2 0 1) 占有順序テーブルファイルをメモリ上のテーブルに読み込む。

40 【 0 0 2 3 】 (ステップ 2 0 2) 占有順序番号の上昇順をチェックするための直前情報退避エリアを初期設定。

【 0 0 2 4 】 (ステップ 2 0 3) ソートされたトレースレコードを 1 件ずつ入力する。

【 0 0 2 5 】 (ステップ 2 0 4) トレースレコード終了まで繰り返す。

【 0 0 2 6 】 (ステップ 2 0 5) トレースレコード上のリソース I D 4 0 6 から、占有順序ルールテーブル (3 0 2) をサーチし、占有順序番号 (3 0 1) を検索し、トレース情報に付加する (図 5 5 0 7) 。

50 【 0 0 2 7 】 (ステップ 2 0 6) トランザクション I D が変わった場合、そのレコードは順序チェックしないの

でステップ209に分岐する。

【0028】（ステップ207）占有順序番号の上昇順をチェックする。

【0029】（ステップ208）上昇順でない場合、警告メッセージを付加する（図5 507）（ステップ209）占有順序番号507、警告メッセージ508を付加したトレース情報レコードを検証結果ファイルに出力する。

【0030】（ステップ210）占有順序番号の上昇順をチェックするための直前情報退避エリアに今のトレース情報レコードを退避する。

【0031】図5に検証結果の例を示す。ルールに違反しているアクセス部分、つまりルールではリソースCC→DDDの順にアクセスすべきところを、その順序が逆になっているアクセスについては、占有順序番号（507）が上昇順になっていない。

【0032】デッドロック要因として、警告メッセージ（508）を付加して出力する。

【0033】

【発明の効果】本発明によれば、以下の効果が期待出来る。

【0034】（1）リソースの占有順序を確認できると共に、順序不正を機械的に検出できる。

【0035】（2）リソースアクセス順序検証ツールはプログラム本体と独立して作成することが可能であり、プログラム本体の開発工程に影響を与えない。

【0036】（3）性能・リソースの面でプログラム本体に負荷をかけないので性能を重視するシステムには最適なデッドロック防止方式である。

【図面の簡単な説明】

【図3】

301	302
占有順序番号	リソースID
1	AAA
2	BBB
3	CCC
4	DDD
:	:

【図1】本発明の実施例のシステム構成図。

【図2】アクセス順序検証ツールのフローチャート。

【図3】アクセス順序ルールを記述したテーブル例を示す図。

【図4】テスト時に取得されるトレース情報の例を示す図。

【図5】アクセス順序検証ツールによる検証結果の例を示す図。

【符号の説明】

- 10 101…業務プログラムをシステムテストする処理装置、102…業務プログラム、103…テスト実行時に出力されるトレース情報、104…検証の入力となるトレース情報、105…占有順序ルールテーブル、106…占有順序検証ツール、107…検証結果出力ファイル、108…占有順序を検証する処理装置、301…占有順序ルールテーブル上の占有順序番号、302…占有順序ルールテーブル上のリソースID、401…トレース情報上のトランザクションID、402…トレース情報上の時刻、403…トレース情報上のモジュール名、404…トレース情報上の関数名、405…トレース情報上の行番号、406…トレース情報上のリソースID、410…トレース情報の検証単位、411…トレース情報上のトランザクションID（ソート後）、501…検証結果出力ファイル上のトランザクションID、506…検証結果出力ファイル上のリソースID、507…検証結果出力ファイル上の占有順序番号（検証ツールで付加）、508…検証結果出力ファイル上の警告メッセージ（検証ツールで付加）、510…検証結果出力ファイル上の検証単位。

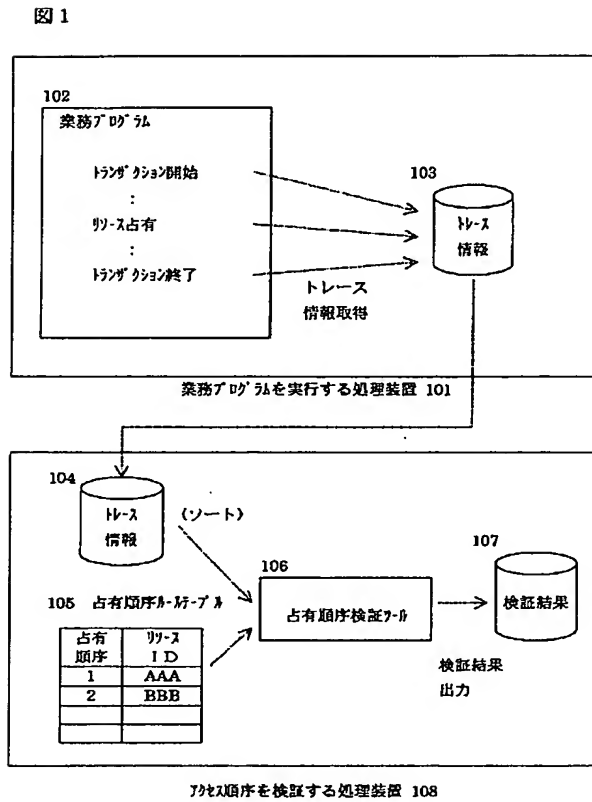
【図5】

図 5

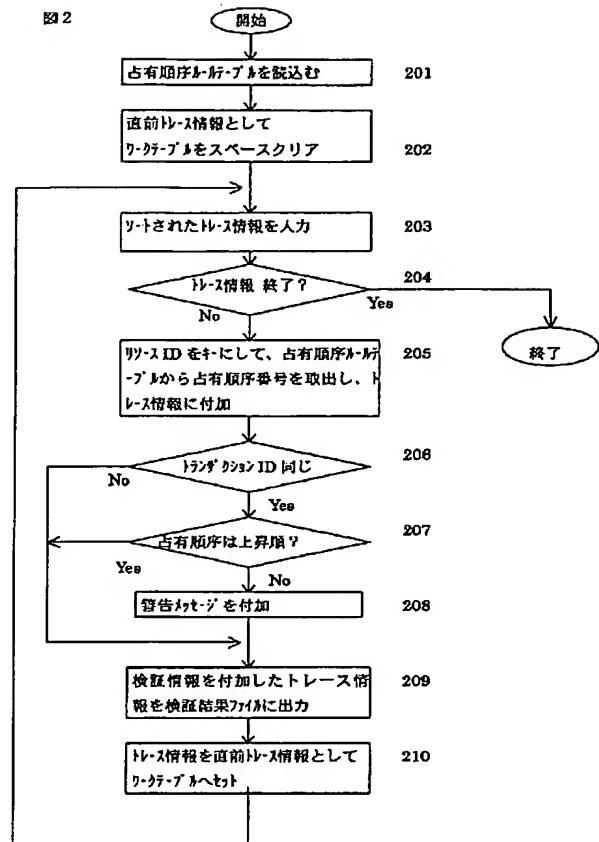
501					506		507		508		510	
トランザク ID	時刻	モジュール名	関数名	行番号	リ ス ID	占有順序 番号	警告 メッセージ					
tttt0001	xxxxxx	XXXX	XXXX	nnnn	AAA	1		↑ 検証単位 ↓				
	xxxxxx	XXXX	XXXX	nnnn	CCC	3						
	xxxxxx	XXXX	XXXX	nnnn	DDD	4						
tttt0002	xxxxxx	XXXX	XXXX	nnnn	AAA	1		↑ 検証単位 ↓				
	xxxxxx	XXXX	XXXX	nnnn	DDD	4						
	xxxxxx	XXXX	XXXX	nnnn	CCC	3	←警告					
tttt0003	xxxxxx	XXXX	XXXX	nnnn	BBB	2						

同じトランザクションIDで
占有順序番号が上昇順でない

【図 1】



【図 2】



【図 4】

図 4

(a) トレース取得時

	401	402	403	404	405	406
トランザクション ID	時刻	モジュール名	関数名	行番号	プロセス ID	
tttt0001	xxxxxx	XXXXX	XXXXX	nnnn	AAA	
tttt0002	xxxxxx	XXXXX	XXXXX	nnnn	AAA	
tttt0003	xxxxxx	XXXXX	XXXXX	nnnn	BBB	
tttt0002	xxxxxx	XXXXX	XXXXX	nnnn	DDD	
tttt0001	xxxxxx	XXXXX	XXXXX	nnnn	CCC	
tttt0001	xxxxxx	XXXXX	XXXXX	nnnn	DDD	
tttt0002	xxxxxx	XXXXX	XXXXX	nnnn	CCC	

(b) 検証のためのソート後

	411					410
トランザクション ID	時刻	モジュール名	関数名	行番号	プロセス ID	
tttt0001	xxxxxx	XXXXX	XXXXX	nnnn	AAA	↑
tttt0001	xxxxxx	XXXXX	XXXXX	nnnn	CCC	検証単位
tttt0001	xxxxxx	XXXXX	XXXXX	nnnn	DDD	↓
tttt0002	xxxxxx	XXXXX	XXXXX	nnnn	AAA	↑
tttt0002	xxxxxx	XXXXX	XXXXX	nnnn	DDD	検証単位
tttt0002	xxxxxx	XXXXX	XXXXX	nnnn	CCC	↓
tttt0003	xxxxxx	XXXXX	XXXXX	nnnn	BBB	

↑

トランザクションIDごとに
時系列な並びにソート